METHOD AND APPARATUS FOR HANDLING FAILURES OF RESOURCE MANAGERS IN A CLUSTERED ENVIRONMENT

CROSS-REFERENCE TO RELATED APPLICATION

[0001] This application is a continuation of copending U.S. utility application entitled, "METHOD AND APPARATUS FOR HANDLING FAILURES OF RESOURCE MANAGERS IN A CLUSTERED ENVIRONMENT," having ser. no. 09/267,032, filed March 11, 1999, now issued as U.S. Patent No. _____, which is entirely incorporated herein by reference.

BACKGROUND OF THE INVENTION

[0002] The invention relates generally to a clustered processing system formed from multiple processor units with fault-tolerant capability. More particularly, the invention relates to a method, and apparatus for implementing that method, for handling, in a fault-tolerant manner, the failure of a resource manager in the context of a transaction executing on the system.

[0003] A useful definition of a transaction is that it is an explicitly delimited operation, or set of related operations, that change or otherwise modify the content of an information collection or database from one consistent state to another. Changes are treated as a single unit in that all changes of a transaction are formed and made permanent (i.e., the transaction is "committed") or none of the changes are made permanent (i.e., the transaction is "aborted"). If a failure occurs during the execution of a transaction, the transaction can be aborted and whatever partial changes were made to the collection can be undone to leave it in a consistent state.

[0004] Typically, transactions are performed under the supervision of a transaction manager facility (TMF). In geographically distributed systems, such as multiple processor unit systems or "clusters" (i.e., a group of independent processor units managed as a single system), the TMF is "distributed" in the sense that each processor unit has its own TMF component to coordinate operations of a transaction conducted on that processor unit. The processor unit at which (or on which) a transaction begins is sometimes called the "beginner" processor. The TMF component of that processor unit operates to coordinate those transactional resources remote from its resident processor unit (i.e., resources managed by other processor units). Those TMF components running on processor units managing resources enlisted in a transaction

are "participants" in the transaction. And, it is the TMF component of the beginner processor unit that initiates the steps taken.

[0005] A preferred approach to concluding the transaction, and confirming that all participant resources employed in a transaction are able to participate in that conclusion, is to use the Two-Phase Commit ("2PC") protocol. According to this approach, the beginner TMF component, upon receipt of an "End Transaction" request from the application process that requested the transaction, broadcasts a "Prepare" signal to all processor units of the cluster. The processor units, upon receipt of the Prepare signal, notify their (local) participant resources to perform as necessary (e.g., completing writes to disk storage, clearing memory, etc.) for effecting the change in state of the database and, if the necessary operation succeeds, respond with a "Ready" signal. If all participants of the transaction respond with an affirmative, i.e., a "Ready" signal (and "Not Involved" signals received from any processor units not participating in the transaction), the beginner TMF component notifies a transaction monitor process (TMP), running on one of the processor units, to "commit" the change to an audit log. The TMP tells the beginner TMF component that the transaction is committed, and the beginner TMF component then broadcasts a "Commit" signal to the participant processor units. At this point the change is considered permanent.

[0006] Fault tolerance is another important feature of transaction processing. Being able to detect and tolerate faults allows the integrity of the information collection being managed by the system to be protected. Although a number of different methods and facilities exist, one particularly effective fault tolerant technique is the "process-pair" technique, as it is sometimes called. (This technique is also sometimes referred to as "fail-over" capability.) According to this technique, an application program is instantiated as two separate processes, a primary process resident on one processor unit of the cluster, and a backup process resident on another processor unit. If the primary process, or the processor unit upon which it is running, fails, that failure brings into operation the backup process to take over the operation of the lost (primary) process. If that failure occurs during a transaction in which the lost process was a participant, the backup decides whether or not to notify the beginner processor unit to abort the transaction and begin over again. In this way the state of the collection managed by the system remains consistent. An example of the process-pair or fail-over technique can be found in U.S. Pat. No. 4,817,091.

[0007] An alternative approach, one used for example by the software applications that use object linking and embedding (OLE), is to create a backup process only after the primary process is detected as having failed. The state needed by the newly-created backup is transferred after creation. One problem with this approach is that the state needed by the backup is often retained by the node or processor unit on which the primary was running. If it happens that the primary process has failed because the processor unit on which it was running failed, or it has lost the capability to communicate with the transaction manager, that state can be lost.

[0008] Also, there are times when the failure of a process, and the subsequent fail-over of the failed process to another processor unit (i.e., to the backup process), tend to impede transactions. For example, a stage in a transaction may be reached such that the participants no longer are able to abort the transaction. Should a participant process, or the processor unit, or some other facility related to the participant process, fail, the transaction will not be committed, and state used by the failed process will be left to clutter the system.

[0009] These problems normally do not occur in a coordinated system having component parts designed to work together. They most often appear when porting an application from one platform to another.

[0010] Accordingly, it can be seen that there exists a need for being able to provide full fail-over capability in a transaction processing system in order to maintain fault-tolerance. It follows, therefore, that the state created and maintained by the primary process should be placed where it can be reached for use by the backup process when necessary, regardless of how the primary process fails or is lost.

SUMMARY OF THE INVENTION

[0011] According to the present invention, in a transaction processing system using a transaction management facility (TMF) a certain state is written to the audit log maintained by the TMF when a point (using a two-phase or 2PC commit operation) in a transaction is reached beyond which participation of the resources used in the transaction are required. Typically, the point is when the Ready signal is received in response to the Prepare signal broadcast by the beginner TMF. According to the invention, the Ready signal is accompanied by state information from which the state needed by that participant can be recreated, and written to an audit log by the beginner TMF. If the participant fails, or is otherwise made unavailable, a backup participant

is created - preferably on another node - and provided with the same identifier of the now-failed participant. The backup participant queries TMF to determine if any transactions are outstanding and associated with the identifier. Responding, the TMF supplies the backup participant with the retained state information previously stored in the audit log, thereby allowing the backup participant to complete as necessary the transaction previously involving the failed participant.

[0012] A significant advantage of the invention is to allow OLE compliant applications, such as the Microsoft SQL Server (e.g., the Microsoft SQL Server 6.5) or the Microsoft Message Queue Server, to be ported to a foreign platform and yet keep their fault tolerant capability which relies upon detection of a failure of a process before a backup of that process is created.

[0013] These and other advantages and aspects of the invention will become apparent to those skilled in this art upon a reading of the following detailed description of the invention, which should be taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] FIG. 1 is an illustrative diagram of a multiple processor cluster or system;

[0015] FIG, 2 is a flow diagram illustrating creation and migration of a resource manager process; and

[0016] FIG. 3 is flow diagram broadly illustrating the steps a conventional two-phase commit protocol modified according to an implementation of the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENT

[0017] The present invention is advantageous in that it permits OLE compliant applications and resources to be ported to a transaction processing system and to participate in transactions with a minimum of re-work of the application or process for the porting process. However, as those skilled in this art will see, the techniques used by the present invention can be readily employed in other systems (i.e., non-OLE systems). The present invention is designed to operate with a transaction processing facility that runs under the aegis of the NonStop operating system available from the assignee of the invention, the Compaq Computer Corporation, Cupertino, California. Thus, the invention allows OLE applications to conduct transactions under the Nonstop operating system Transaction Manager Facility (TMF) and/or use such OLE compliant resource manager processes as Microsoft SQL Server 6.5 (a high

performance database management system for Windows NT-based systems) or the Microsoft Message Queue Server (a transactional support facility that allows message queues to participate in Microsoft Transaction Server transactions). These resource managers are constructed to operate in the context of another transaction manager. The present invention allows their use in the context of a foreign transaction manager and foreign operating system, yet still employ the fault tolerant techniques originally designed for them. (Microsoft, Microsoft Windows NT are registered trademarks of Microsoft Corporation of Redmond, Washington, and Microsoft SQL Server, and Microsoft Transnational Server are believed to be trademarks of Microsoft Corporation.)

[0018] Turning now to the figures, and for the present FIG. 1, there is illustrated a multiple processor transaction processing system 10 capable of employing the present invention. As FIG, 1 shows, the transaction processing system 10 includes central processor units (CPUs) 12 (12a, 12b, ..., 12d) and peripheral devices 14 (14a, 14b, ..., 14d) interconnected by a communication fabric 14 that provides both interprocessor and input/output (I/O) communication. Preferably, the communication fabric 14 is constructed as is taught in U.S. Pat. No. 5,751,932. However, as will be evident to those skilled in this art, other multiprocessor architectures may be used, such as that taught in U.S. Pat. No. 4,228,496. As will also be evident to those skilled in this art, although only four CPUs are shown, the present invention may be used on any number of CPUs.

[0019] As conventional, the transaction processing system 10 also includes the necessary hardware, software, procedures, rules, and users needed to implement and operate a transaction processing application, including the NonStop or other operating system. In addition, there is a distributed transaction manager facility (TMF), comprising a transaction manager process (TMP) 24 resident on one of the CPUs 12 (in FIG. 1, CPU 12c), and TMF components 26 each allocated to an individual processor 12; that is, each of the processors 12 has a TMF component 26 (26a, 26b, ..., 26n) that operates to manage and track the local resource managers (RMs) running on that CPU (e.g., RM(1) on CPU 12b or RM(2) on CPU 12d). (The resource manager RM(2) is shown in phantom because, as will be seen in connection with the discussion below, it is created later as a backup to the resource manager RM(1).)

[0020] Preferably, the system 10 includes a fail-over capability, such as that provided by the NonStop operating system in the form of the "process pair" technique

discussed above. However, for the OLE compliant processes there is provided a Microsoft Cluster Service (MSCS) 28. FIG. 1 also shows each CPU 12 having a component of MSCS 28. MSCS 28 operates to provide a fault tolerance capability by detecting failure of a process, or CPU 12 on which a process is running, and creating replacement or backup process on another CPU for taking over the function and operation of the failed process. As will be seen, the present invention provides the means to allow this "failover" to take place with a minimum of effort.

[0021] Since applications and resource managers, including those that are the OLE compliant, need a resource that provides an efficient communication link to TMF 26 there is provided a distributed transaction coordinator gateway (DTC-GW) 30. Although there needs to be only one DTC-GW 30 running on one of the CPUs 12, it is more efficient to have each CPU 12 to include at least one DTC-GW as FIG. 1 illustrates. Not only is the efficiency improved (by avoiding having processes use a DTC-GW on a CPU 12 different from its own), but, as will be seen, the failover capability provided by the MCSC 28 for fault tolerance is made more effective.

[0022] Finally, communication between the applications (e.g., APP 29) and resource managers (e.g., RM(1), R.M(2)) and the DTC-GW 30 of the particular CPU 12 is via a driver 32 (32a, 32b, ..., 32d). In addition, each DTC-GW 30 is communicatively coupled to the TMF component 26 of its particular CPU 12.

[0023] FIG. 2 shows a flow chart 40 that broadly illustrates the steps taken to create a process such as RM(l). Accordingly, when RM(1) is created, a dynamic-linked library (DLL) is loaded in step 42 to provide the RM(1) with various interfaces (e.g., procedure calls) required to communicate with the local transaction manager. Next, at step 44, the RM(1) is assigned a globally unique identifier (GUID). This GUID uniquely identifies the resource to the TMF 26, distinguishing it from all other processes (e.g., APP 29). Then, at step 46, a connection, including a driver 32, is provided the resource to the local DTC-GW 30. RM(1) then, at step 48, queries TMF 26 to determine if there is any transaction it should be aware of. Since the resource has been created as a primary resource, there is no such transaction. If, on the other hand (as discussed further below) the RM(l) was created to replace a failed resource, and that failed resource was a participant in a transaction when it failed, TMF 26 could respond in the affirmative. This feature is discussed further below.

[0024] When a transaction is started in one CPU 12, that CPU 12 is known as the "beginner" CPU, and the TMF component 26 of that CPU becomes the "beginner"

TMF component. If the transaction involves an operation performed on or at a CPU 12 other than the beginner CPU 12, that other CPU and its TMF component 26 become "participants" of the transaction and subordinate to the beginner TMF component on the beginner CPU. This may be better understood with an example.

[0025] Assume the APP 29 is requested to perform some operation or operations that requires the state of a database to be changed, and to perform that operation or operations the application (APP) 29 must use the resource(s) managed by resource managers of the system 10 such as RM(1). The APP 29 makes a "Start Transaction" call to its local TMF component 26a to register the transaction The TMF component 26a (now, the beginner TMF component), by this call (as is conventional), receives the information it needs to track the transaction so that it can ensure that the transaction completes properly. To enlist the services of the resource manager RM(1), (probably in another CPU) APP 29 sends a request for the resource manager RM(1) to modify the database maintained by the system 10. When RM(1) receives this request, it first contacts its local DTC-GW to notify its local TMF component 26b that it is a participant in the transaction started by the APP 29. The DTC-GW first opens a logical connection between it and the TMF 26 for this transaction, and communicates the notification from the RM(1). All communication with the TMF components by the APP 29 and the RM(1) is through the local DTC-GW 30 (i.e., DTC-GW 30a and 30b).

[0026] FIG. 3 broadly illustrates, by the flow diagram 60, the major steps taken to make permanent the change or modification. When the request for work has been sent by the APP 29, and APP 29 has nothing else to do, it then makes, in step 62, an "End Transaction" call to the beginner TMF component 26a The beginner TMF component 26a, in turn, performs the necessary operations to make the change or modification permanent and consistent. Preferably, the conventional two-phase commit (presumed abort) protocol is used in which, at step 64, the beginner TMF component 26a broadcasts a "Prepare" signal to all CPUs 12. Those CPUs 12 having resource manager participants in the transaction -- here RM(1) -- perform as necessary (e.g., completing writes to disk storage) for effecting the change in state of the database and, if the necessary operation succeeds, at step 68, respond with a "Ready" signal. If any participant responds with an "Abort" signal (step 70), or one or more participants fail to respond with the obligatory Ready signal (step 74), the procedure 60 proceeds to step 72 to initiate a rollback of the transaction.

[0027] If, on the other hand, there are no Abort signals (step 70); and all participants of the transaction respond with an affirmative, i.e., a "Ready" signal (step 74 and "Not Involved" signals received from any CPUs 12 not participating in the transaction) the beginner TMF component 26a passing from step 74 to step 80 (bypassing here, for the moment, step 76) notifies the TMP 24 to "commit" the change to an audit log. The TMP 24 tells the beginner TMF component 26b that the transaction is committed. The beginner TMF component 26b, at step 80, then broadcasts a "Commit" signal to the participant CPUs 12. At this point the change is considered permanent. Upon receipt of the Commit signal, the RM(1) cleans up whatever state is left from the operation(s) it performed in connection with the transaction.

[0028] Suppose, however, that during the transaction the RM(1) fails, or the communicative connection to its associated JJTC-GW 30b is lost. If this occurs before the Ready signal (step 68) is received from the RM(1), beginner TMF assumes (correctly) that either the CPU 12b or the RM(1) has failed and abort the transaction. However, if the failure occurs after the Ready signal is received, beginner TMF can commit the transaction without knowing that RM(1) is unable to at least cleanup its state and complete the necessary operations. Even if a replacement is created for RM(1) by the MSCS 28 in the form of the RM(2) on CPU 12d, RM(2) cannot complete what needs be done because the state is often located with the original CPU in the associated DTC-GW 30b. The reason is that the necessary state associated with the RM(1) was not available to the replacement resource manager in prior systems because it is usually kept by the DTC-GW 30.

[0029] Thus, according to the present invention, the conventional 2PC procedure, as described above, is modified to include step 76. To that end, when all participants respond with Ready signals, the TMF components receive, at step 68, state information from the corresponding DTC-GW 30, piggy-backed on the Ready signal. Then, at step 76, the state information is written to an audit log (not shown).

[0030] Now, assume that after the RM(1) sends the responsive ready signal, accompanied by state information respecting the transaction to which the Ready signal pertains, the communicative connection P14 (1) enjoyed with the DTC-GW 30b fails, or RM(1) itself fails, or the CPU 12b fails, the MSCS is then apprised of either of these facts and, in turn, notifies TMF 26. TMF 26, in turn, assumes that the associated logical connections for the P14(1) have been closed. This is necessary, as

will be seen, to allow TMF 26 to respond to queries from resource managers identifying themselves with the same GUID as that used by the RM(1).

[0031] Next, the MSCS 28 creates a copy of RM(1), RM(2), on the CPU 12d (FIG. 1) as a backup, following the same steps illustrated in FIG. 2 and described above. When the P14(2) is up and running, it (step 46; FIG. 2) establishes a connection, through a driver 32, with the local DTC-GW, DTC-GW 30d in order to be able to communicate with TMF (i.e., the TMF 26d component for that CPU). Then, in step 48, the backup resource manager to RM(1), RM(2), queries the TMF 26d component, using the GUID formally identifying the RM(1), in effect asking if there are any outstanding transactions in which RM(1) was a participant. Here, there is, assuming RM(1) was lost after RM(1) sent its Ready signal, but before it received the Commit signal. Accordingly, TMF 26d responds in the affirmative by accessing the audit log for the state information originally sent it with the Ready signal from the RM(1), recreates that state from the state information, and forwards it to RM (2). RM(2) sees that it (through RM(1)) is a participant in a transaction corresponding to the state data it received, and queries TMF 26d about that transaction. TMF 26d, replies, telling RM(2) that it has committed the transaction. RM(2) then takes steps to commit the transaction.